Sprawozdanie 1, MPI

Metody programowania równoległego

Autor: Paweł Mendroch

1. Kody programów

Ex1: Bandwidth test z użyciem funkcji **MPI_Send** oraz **MPI_Recv**.

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
  MPI_Init(NULL, NULL);
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  int arraySize = 10000000;
  int sendTimes = 100;
  int mBit = 125000;

  if (world_size < 2)
  {
    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }
  if (world_rank == 0)
  {
    int i;
    int *dataArray = malloc(arraySize * sizeof(int));
    for (i = 0; i < arraySize; i++)
    {
      dataArray[i] = 9;
    }
```

```
    MPI_Barrier(MPI_COMM_WORLD);
    for (i = 0; i < sendTimes; i++)
    {
      double start = MPI_Wtime();
      MPI_Send(dataArray, arraySize, MPI_INT, 1, 0, MPI_COMM_WORLD);
      double end = MPI_Wtime();
      double timeElapsed = end - start;
      printf("%f\n", sizeof(dataArray) / timeElapsed / mBit);
    }
  }
  else if (world_rank == 1)
  {
    int i;
    int *dataArray = malloc(arraySize * sizeof(int));

    MPI_Barrier(MPI_COMM_WORLD);

    for (i = 0; i < sendTimes; i++)
    {
      MPI_Recv(dataArray, arraySize, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    }
  }
  MPI_Finalize();
}
```

Ex2: Ping test z użyciem funkcji **MPI_Send** oraz **MPI_Recv**.

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
  MPI_Init(NULL, NULL);
```

```
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
  int i;
  int sendTimes = 100;

  if (world_size < 2)
  {
    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }
  if (world_rank == 0)
  {
    int number = -1;
    MPI_Barrier(MPI_COMM_WORLD);
    for (i = 0; i < sendTimes; i++)
    {
      double start = MPI_Wtime();
      MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
      double end = MPI_Wtime();
      printf("%f\n", (end - start) * 1000 / sendTimes);
    }
  }
  else if (world_rank == 1)
  {
    int number;
    MPI_Barrier(MPI_COMM_WORLD);
    for (i = 0; i < sendTimes; i++)
    {
      MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
  }
  MPI_Finalize();
}
```

Ex3: Bandwidth test z użyciem funkcji **MPI_Ssend** oraz **MPI_Recv**.

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
  MPI_Init(NULL, NULL);
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  int arraySize = 1000;
  int sendTimes = 100;
  int mBit = 125000;

  if (world_size < 2)
  {
    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }
  if (world_rank == 0)
  {
    int i;
    int *dataArray = malloc(arraySize * sizeof(int));
    for (i = 0; i < arraySize; i++)
    {
      dataArray[i] = 9;
    }

    MPI_Barrier(MPI_COMM_WORLD);
    for (i = 0; i < sendTimes; i++)
    {
      double start = MPI_Wtime();
```

```
      MPI_Ssend(dataArray, arraySize, MPI_INT, 1, 0, MPI_COMM_WORLD);
      double end = MPI_Wtime();
      double timeElapsed = end - start;
      printf("%f\n", sizeof(dataArray) / timeElapsed / mBit);
    }
  }
  else if (world_rank == 1)
  {
    int i;
    int *dataArray = malloc(arraySize * sizeof(int));

    MPI_Barrier(MPI_COMM_WORLD);

    for (i = 0; i < sendTimes; i++)
    {
      MPI_Recv(dataArray, arraySize, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    }
  }
  MPI_Finalize();
}
```

Ex4: Ping test z użyciem funkcji **MPI_Ssend** oraz **MPI_Recv**.

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
  MPI_Init(NULL, NULL);
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
  int i;
```

```
  int sendTimes = 100;

  if (world_size < 2)
  {
    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }
  if (world_rank == 0)
  {
    int number = -1;
    MPI_Barrier(MPI_COMM_WORLD);
    for (i = 0; i < sendTimes; i++)
    {
      double start = MPI_Wtime();
      MPI_Ssend(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
      double end = MPI_Wtime();
      printf("%f\n", (end - start) * 1000 / sendTimes);
    }
  }
  else if (world_rank == 1)
  {
    int number;
    MPI_Barrier(MPI_COMM_WORLD);
    for (i = 0; i < sendTimes; i++)
    {
      MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
  }
  MPI_Finalize();
}
```

2. Dane pomiarowe

Testy były uruchamiane na 4 różnych konfiguracjach *machinefile*:

- vnode-01.dydaktyka.icsr.agh.edu.pl:1

- vnode-01.dydaktyka.icsr.agh.edu.pl:2

- vnode-01.dydaktyka.icsr.agh.edu.pl:1
  vnode-02.dydaktyka.icsr.agh.edu.pl:1

- vnode-05.dydaktyka.icsr.agh.edu.pl
  vnode-06.dydaktyka.icsr.agh.edu.pl

Pingi były testowane dla przesyłu zwykłego integera, powtórzone stukrotnie, a następnie wyniki zostały uśrednione.

Bandwidth test został natomiast wykonany dla przesyłu tablicy integerów o rozmiarach: 1000, 10000, 100000, 1000000, 10000000. Dane również przesyłane stukrotnie i wyniki uśrednione. Wykresy zostały pokazane w skali logarytmicznej.

3. Pomiary

Ping 1:

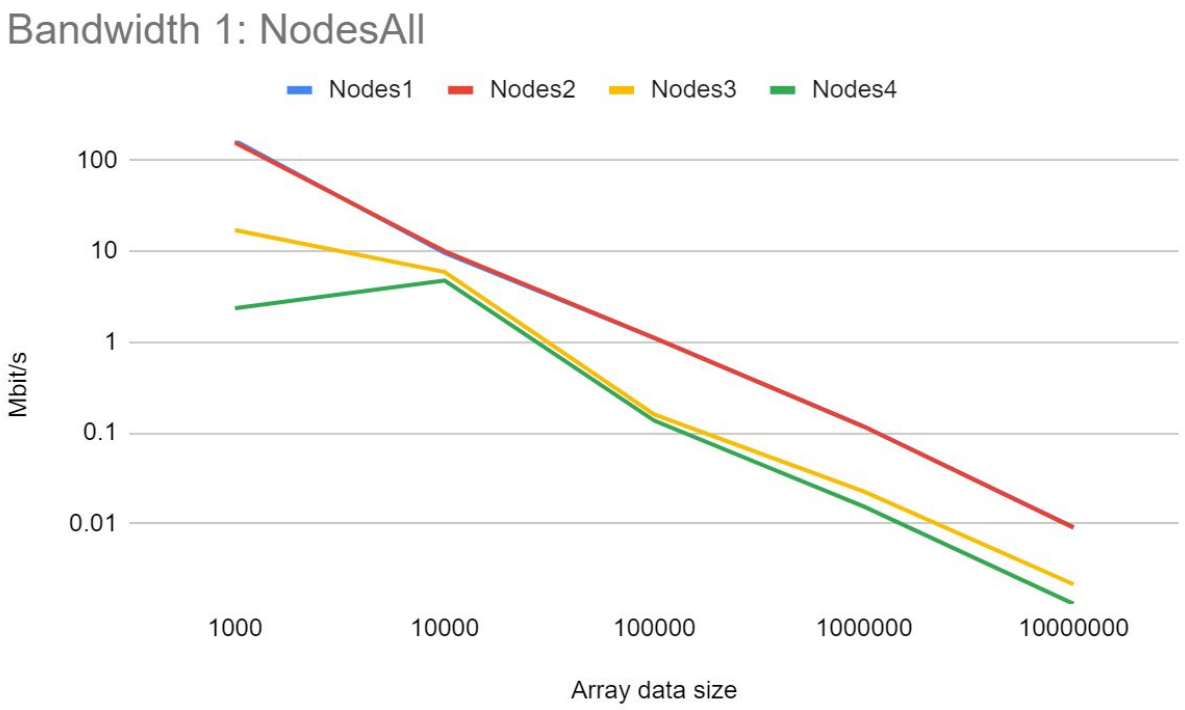
Nodes1: 0.000001 ms
Nodes2: 0.000001 ms
Nodes3: 0.000029 ms
Nodes4: 0.000257 ms

Ping 2:

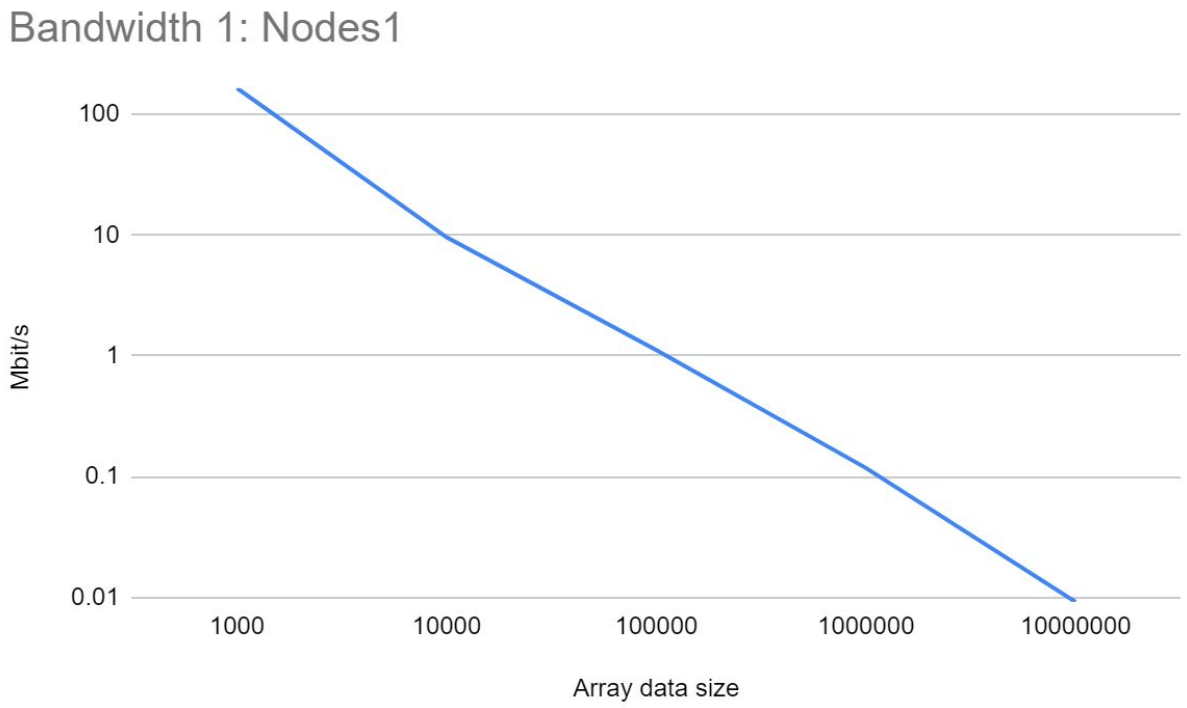
Nodes1: 0.000006 ms
Nodes2: 0.000006 ms
Nodes3: 0.000667 ms
Nodes4: 0.000650 ms

Bandwidth 1:

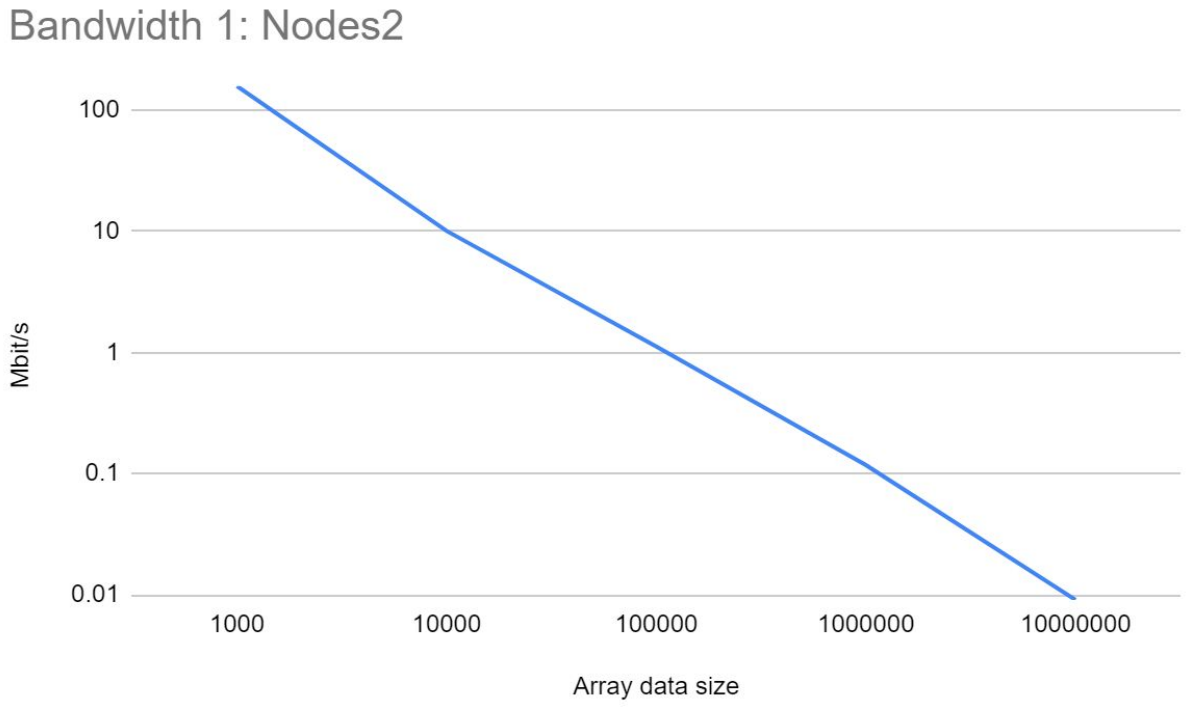
Nodes1&2 (niebieski&czerwony) prawie identyczne pokrycie



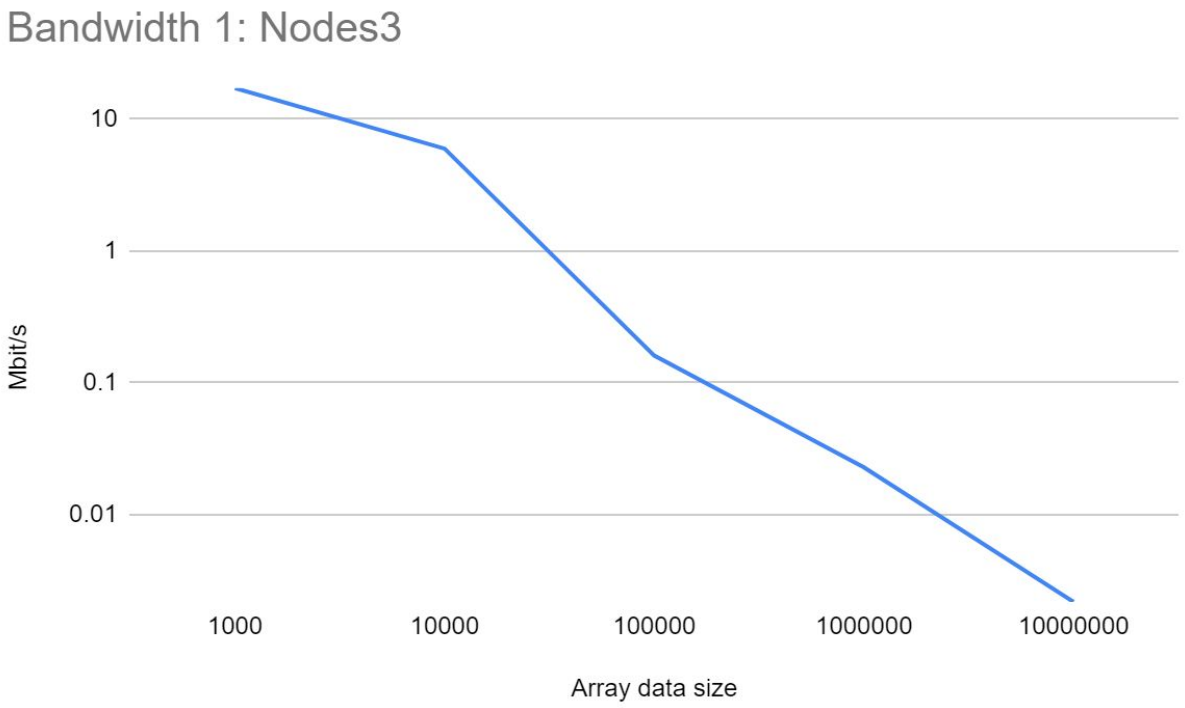
Nodes1:



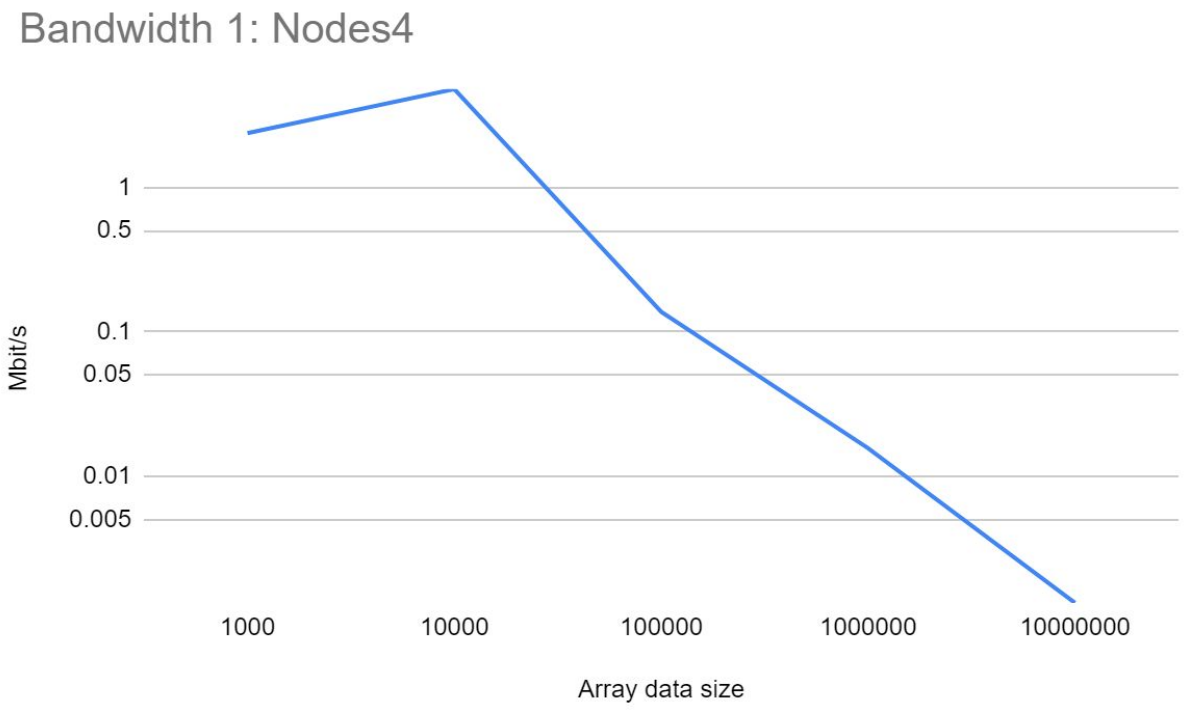
Nodes2:



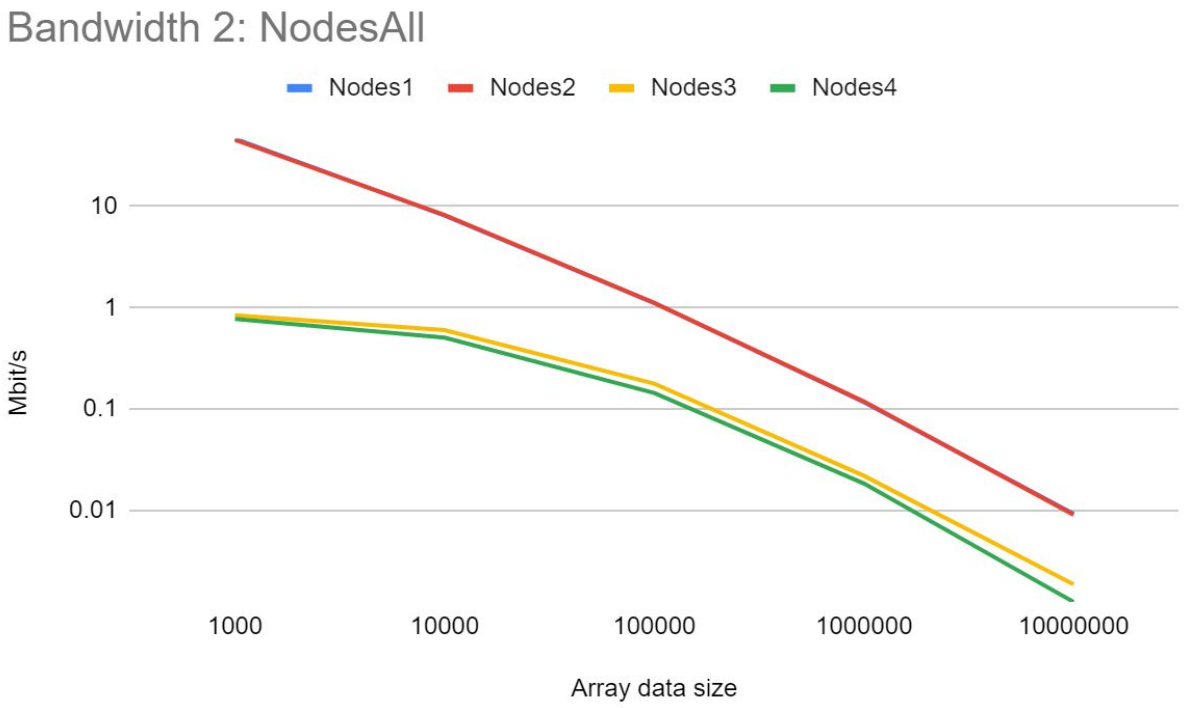
Nodes3:



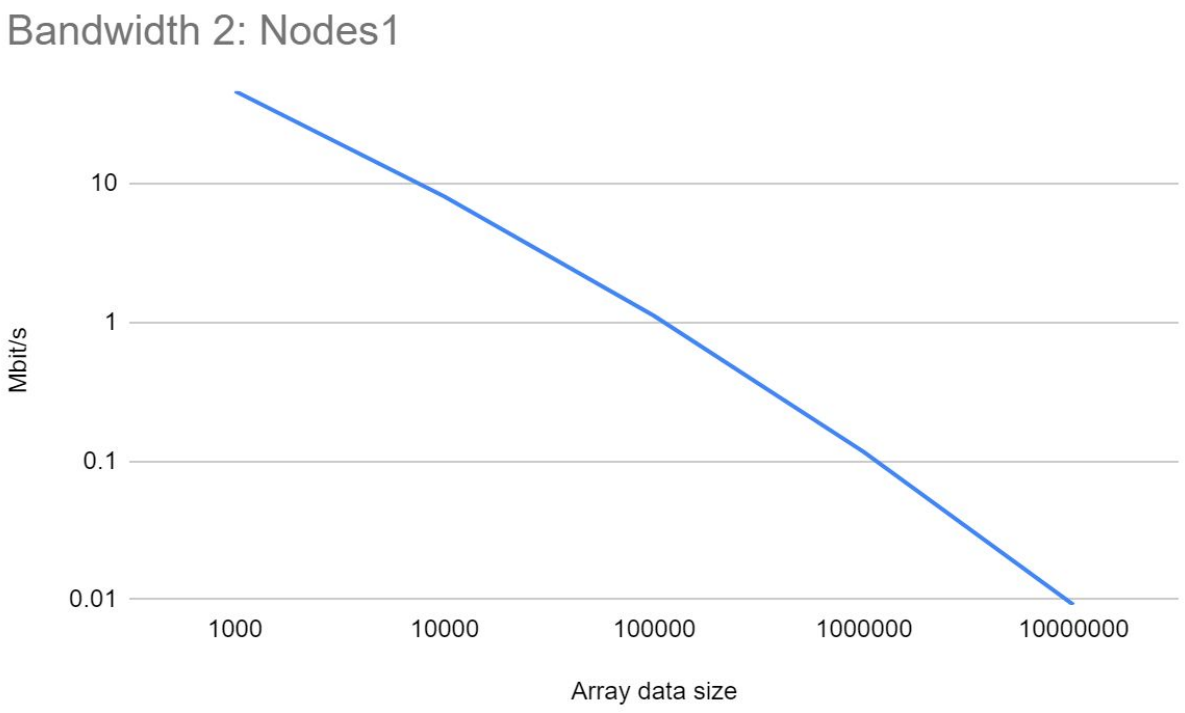
Nodes4:



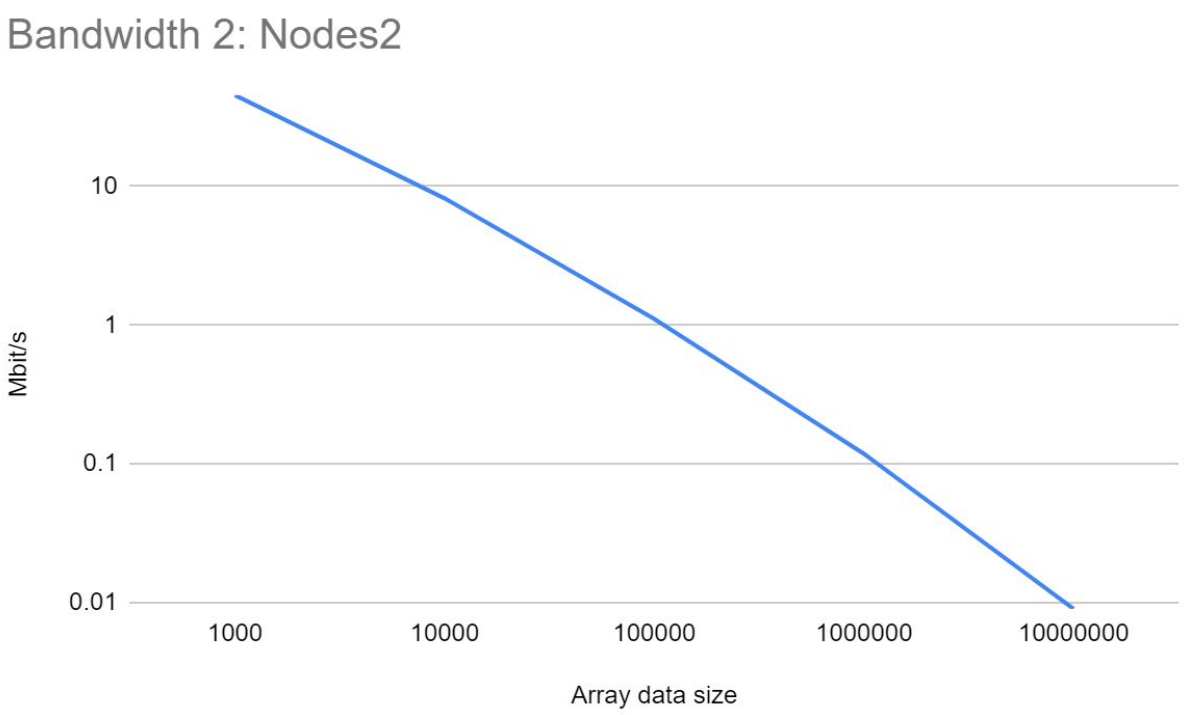
Bandwidth 2:



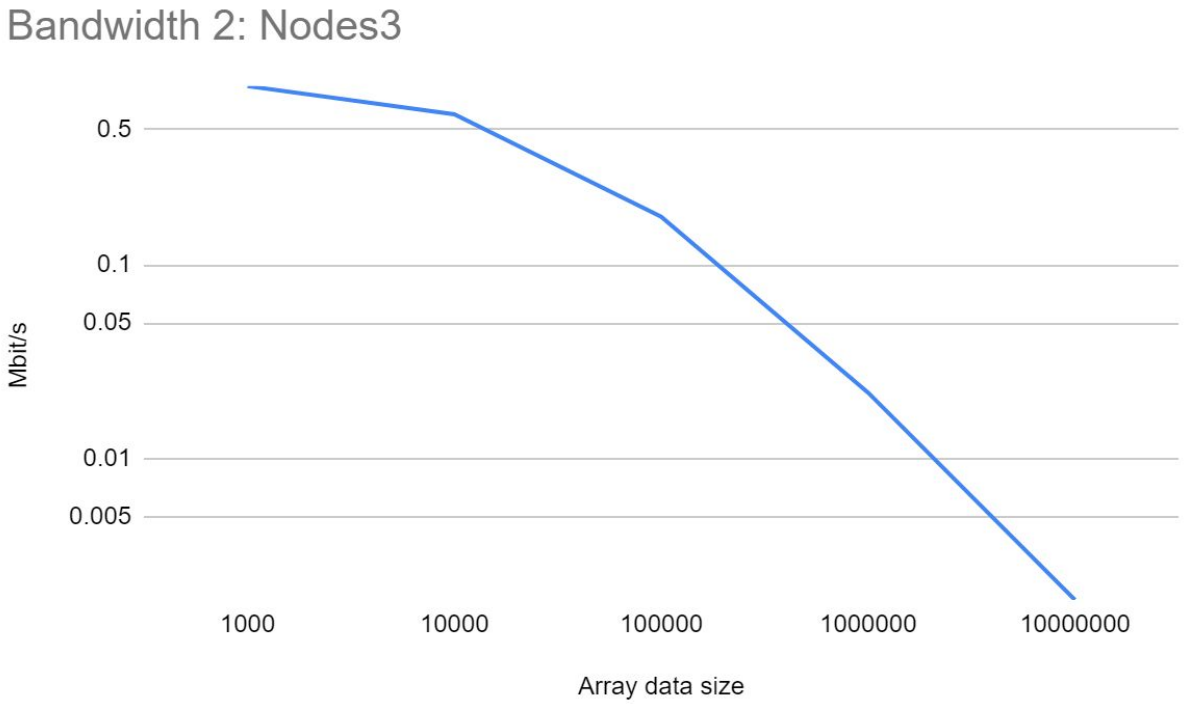
Nodes1:



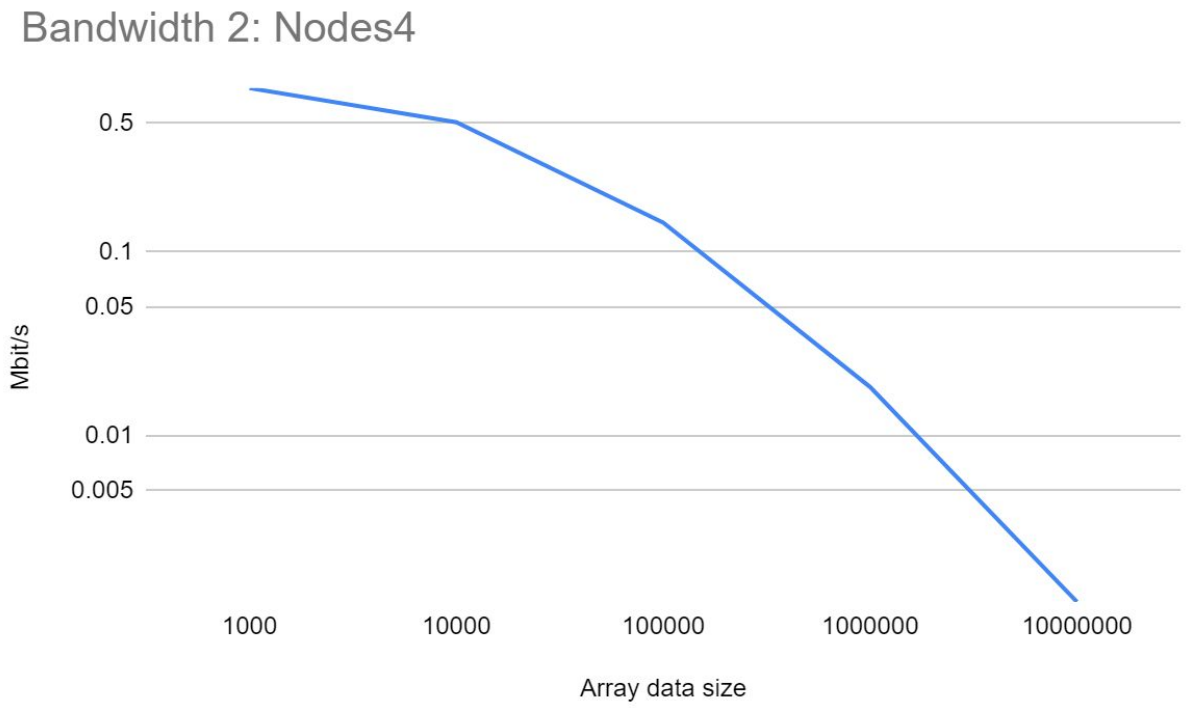
Nodes2:



Nodes3:



Nodes4:


Bandwidth 2: Nodes4
0.5
0.1
0.05
0.01
0.005
Mbit/s
1000
10000
100000
1000000
10000000
Array data size

4. Wnioski

Wyniki były praktycznie identyczne dla przypadku 1 i 2 uruchamiania testów na 1 lub 2 procesorach tego samego node'a 4-procesorowego. Znaczące różnice natomiast rozpoczynały się przy przejściu na różne node'y czy to typu 4-procesorowego czy pojedyncze.

Testy pingów pokazywały ogromne różnice w wykonywaniu programu na tej samej maszynie, a różnych: 0.000001-6 ms, a 0.00025-65 ms. Co ciekawe natomiast przy przesyłaniu standardowym różnica w pingu między setupem Nodes3, a Nodes4 różniła się o jeden rząd dziesiętny, gdzie dla synchronicznego MPI_Ssend wyniki były praktycznie identyczne.

Test bandwidth natomiast pokazał różnice na skali większego przesyłu danych. Tutaj ponownie uruchamianie programu na tym samym procesorze lub na dwóch różnych tej samej maszyny dawało praktycznie te same wyniki, przez co ciężko nawet na wykresie rozróżnić te linie. Korzystanie w dwóch różnych maszyn wieloprocesorowych dawało lepsze wyniki niż korzystanie z dwóch jednoprocesorowych, lecz nieznaczne.
Wszędzie również zauważamy trend spadkowy wraz ze wzrostem ilości danych, gdzie przy rozmiarze 1000 wyniki były odrobinę niestabilne - dokładniejsze obliczenia dawało przesyłanie 100000+ elementów

Zauważamy również, że synchronizacja przesyłu znacząco spowalnia działanie, lecz dopiero przy mniejszej ilości danych. Dla tablicy 1000-elementowej ~150Mbit/s, a ~45Mbit/s dla tej samej maszyny, a dla większej ilości różnice te znacznie malały i prędkość przesyłu stawała się podobna.

Przeprowadzone badanie wykazało spodziewane działanie i wyniki i pozwoliło spojrzeć na faktyczne opóźnienia i możliwości przesyłu w zależności od sposobu komunikacji równoległości.